


Compilation Semantics for a Programming Language with Versions

 Yudai Tanabe¹⁾, Luthfan Anshar Lubis²⁾,
Tomoyuki Aotani³⁾, Hidehiko Masuhara²⁾

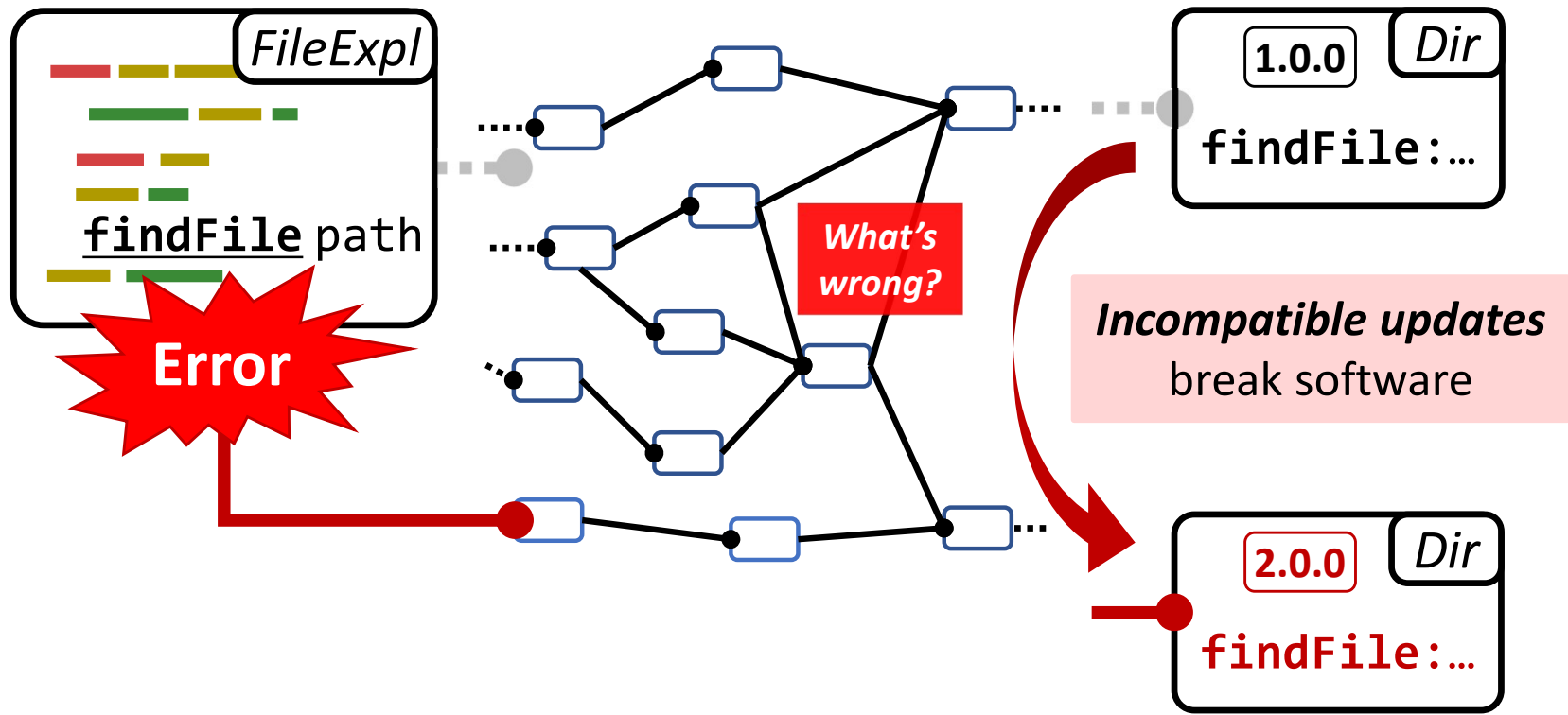
¹⁾Kyoto University, ²⁾Tokyo Institute of Technology, ³⁾Sanyo-Onoda City University

Update Dilemma:

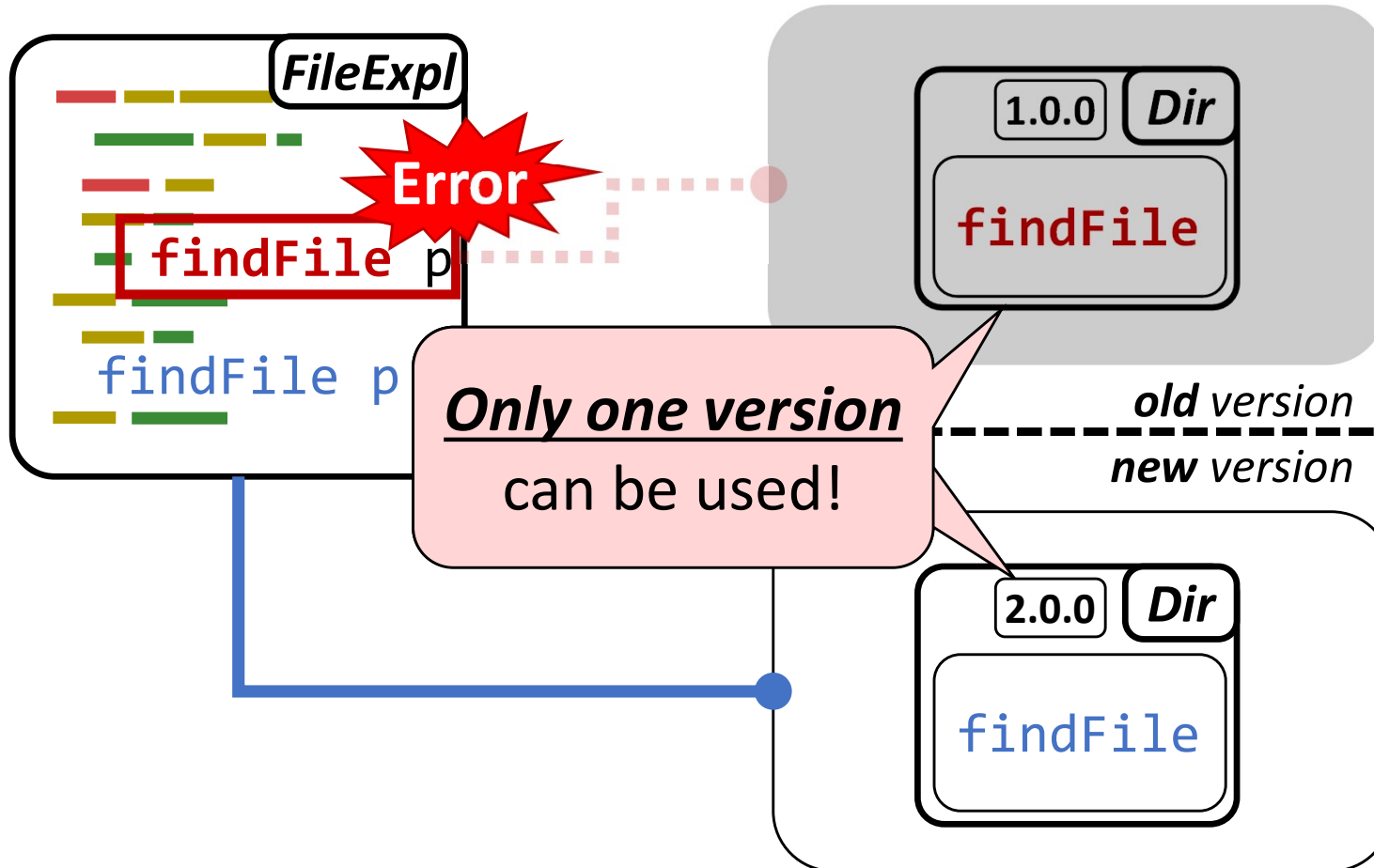
Enhancements vs. Adaptation Costs

[Werner'13, Bavota'15]

Intricate update process deterring programmers from updates

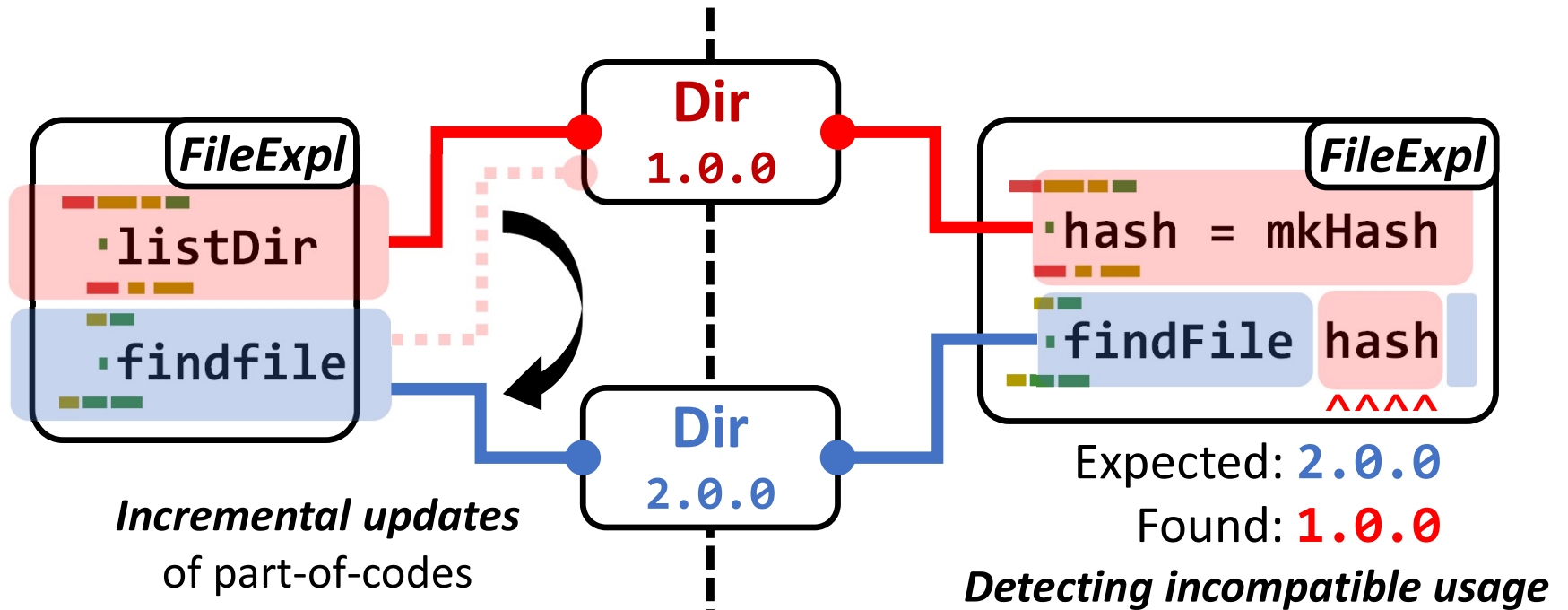


One-version-at-a-time Limitation



Programming With Versions (PWV)

Safely handle multiple versions in one client
to split updates into ***smaller, consistent*** tasks



Basis of this research

FP: λ_{VL} [<Programming>'22],

OOP: BatakJava [SLE'22]

PWV w/o Version Annotations

[<Programming>'22]

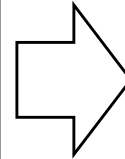
λ_{VL}

This research

VL

```
module FileExpl where

main () =
  let [str] = [getArg [()]] in
  let [digest] =
      [{l1=..., l2=...}[str]] in
  if [{l1=..., l2=...} [digest]].l1
  ...
  [listDir [curDir]].l2
```



```
module FileExpl where

main () =
  let str = getArg () in
  let digest = mkHash str in
  if exist digest ...

  ...
  listDir curDir
```

*Cumbersome
syntax*

*Require versions
in code locations*

**No version
annotations!**

Outline

Contribution

Programming with Versions *w/o* Version Annotations

[<Programming>'22]

λ_{VL}

Explicit
version annotations

- λ_{VL} Semantics and Type System

vs.

This research

VL

Version inference
incorporating implicit versions

- **Key idea:**
Utilizing module versions for expression versions
- Programming in VL
- Compilation Overview

Outline

Contribution

Programming with Versions *w/o* Version Annotations

[<Programming>'22]

λ_{VL}

Explicit
version annotations

- λ_{VL} Semantics
and Type System

vs.

This research

VL

Version inference
incorporating implicit versions

- Key idea:
Utilizing module versions
for expression versions
- VL Programming
- Compilation Overview

λ_{VL} , Versions within Semantics

Version Labels to capture version possibilities

e.g.

$$l_1 = \left[\begin{array}{l} Dir \mapsto 1.0.0, \\ Hash \mapsto 1.0.0 \end{array} \right], l_2 = \left[\begin{array}{l} Dir \mapsto 1.0.0, \\ Hash \mapsto 2.0.0 \end{array} \right]$$

Multiple terms in a **versioned value**

findFile =

$$\left. \begin{array}{l} l_1 = \boxed{\begin{array}{l} \backslash hash \rightarrow \\ \text{if exist hash ...} \end{array}} \\ l_2 = \boxed{\begin{array}{l} \backslash hash \rightarrow \\ \text{if exist hash ...} \end{array}} \end{array} \right\}$$

Evaluate term *in a specific version*

$[findFile \ hash].l_1$

$\rightarrow findFile_{l_1} \ hash_{l_1}$

$\rightarrow /home/yudaitnb$
 $\quad \quad \quad /v1/src/file.ext$

λ_{VL} Type System*Type system to enforce version consistency*

$mkHash : \square_{\{l_1, l_2\}} \text{Hash}$
 $findFile : \square_{\{l_1\}} (\text{Hash} \rightarrow A)$

Denotes *available versions* of a term

 \vdash

let $[f] = findFile$ in
 let $[x] = mkHash$ in
 $[f\ x].l_2$

Inconsistent!

because $l_2 \notin \{l_1\}$

~~Well-typed?~~

Proved

Soundness

$\Gamma \vdash t : A \wedge t \rightarrow t' \Rightarrow \Gamma \vdash t' : A$ (preservation)

$\emptyset \vdash t : A \Rightarrow \text{value } t \vee \exists t'. t \rightarrow t'$ (progress)

Type system is based on coeffect calculi:

ℓRPCF ^[Brunel'14], GrMini ^[Orchard'19].

Outline

Contribution

Programming with Versions *w/o* Version Annotations

[«Programming»'22]

λ_{VL}

Explicit
version annotations

- λ_{VL} Semantics
and Type System

vs.

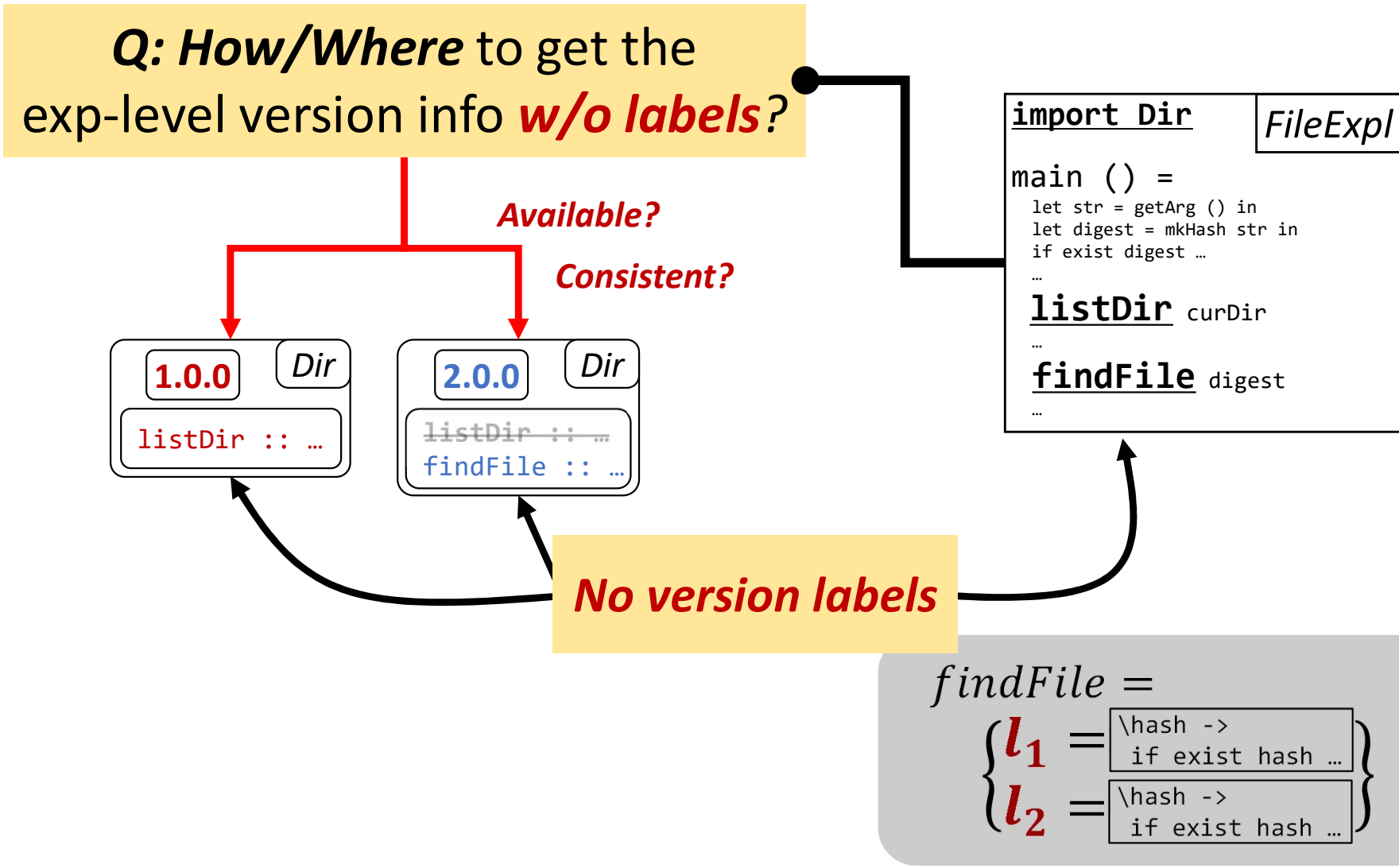
This research

VL

Version inference
incorporating implicit versions

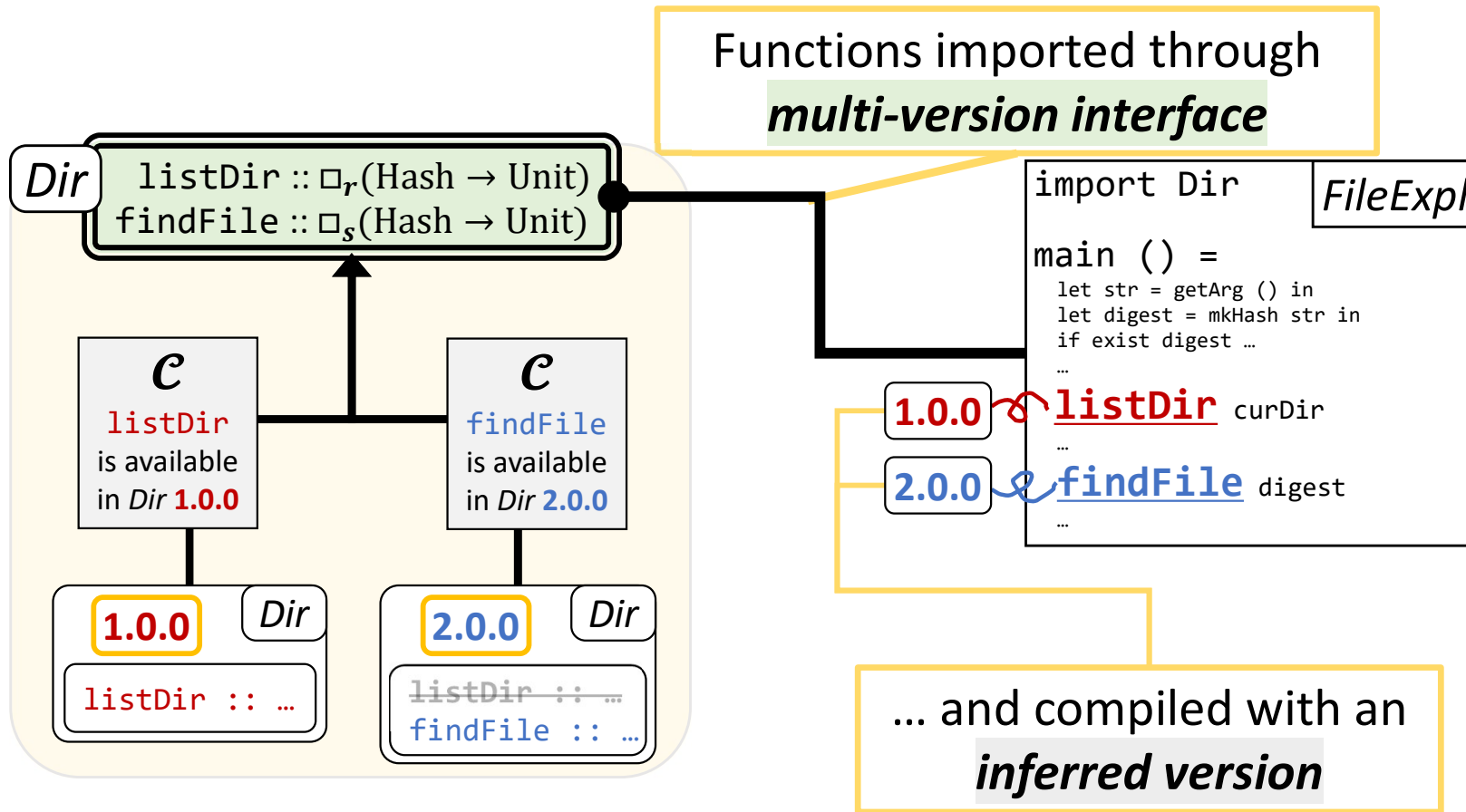
- **Key idea:**
Utilizing module versions
for expression versions
- VL Programming
- Compilation Overview

How to Omit Version Annotations?

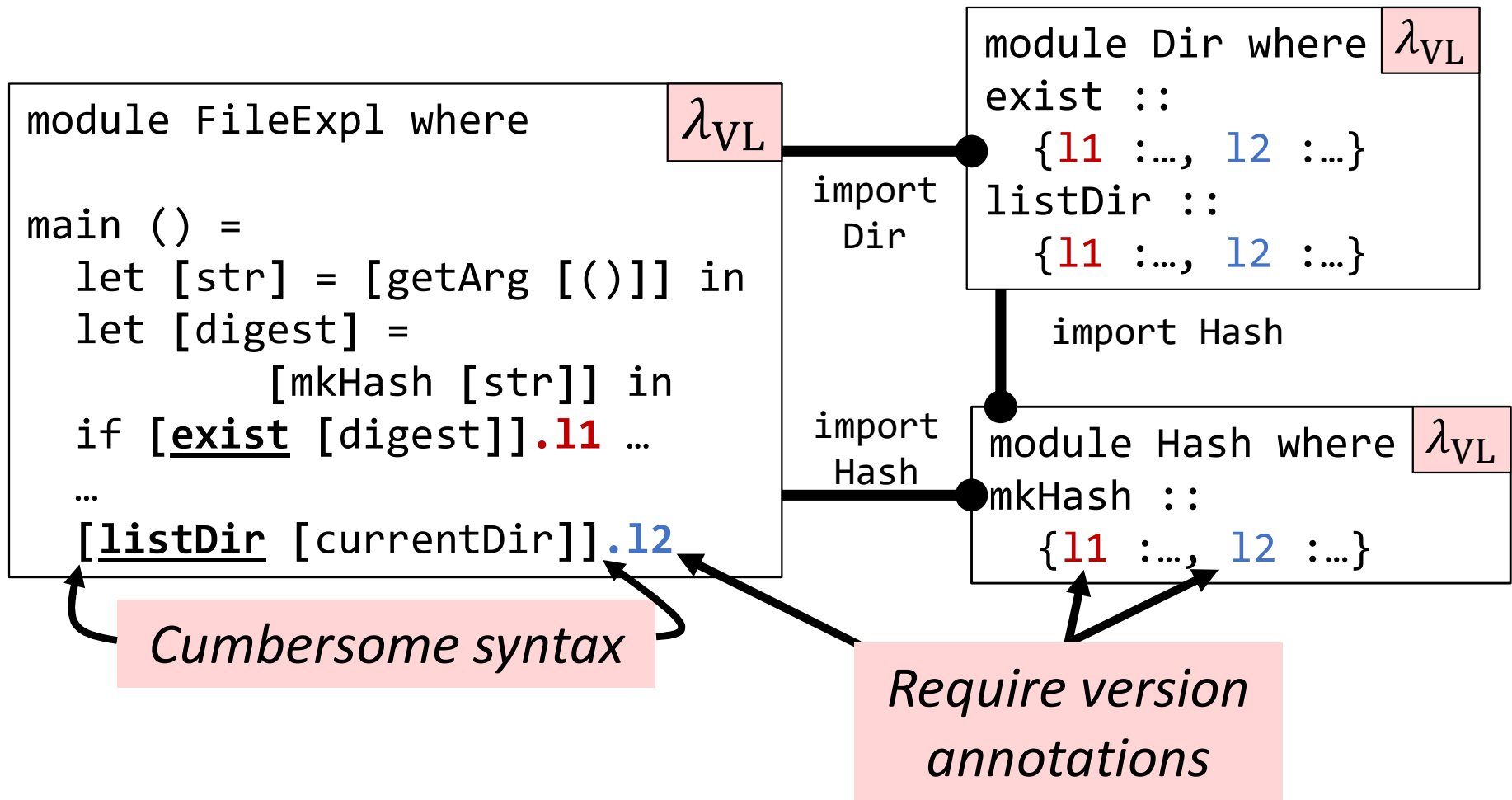


Compilation with *Implicit* Versions

A. Utilizing module vers. to denote expression vers.

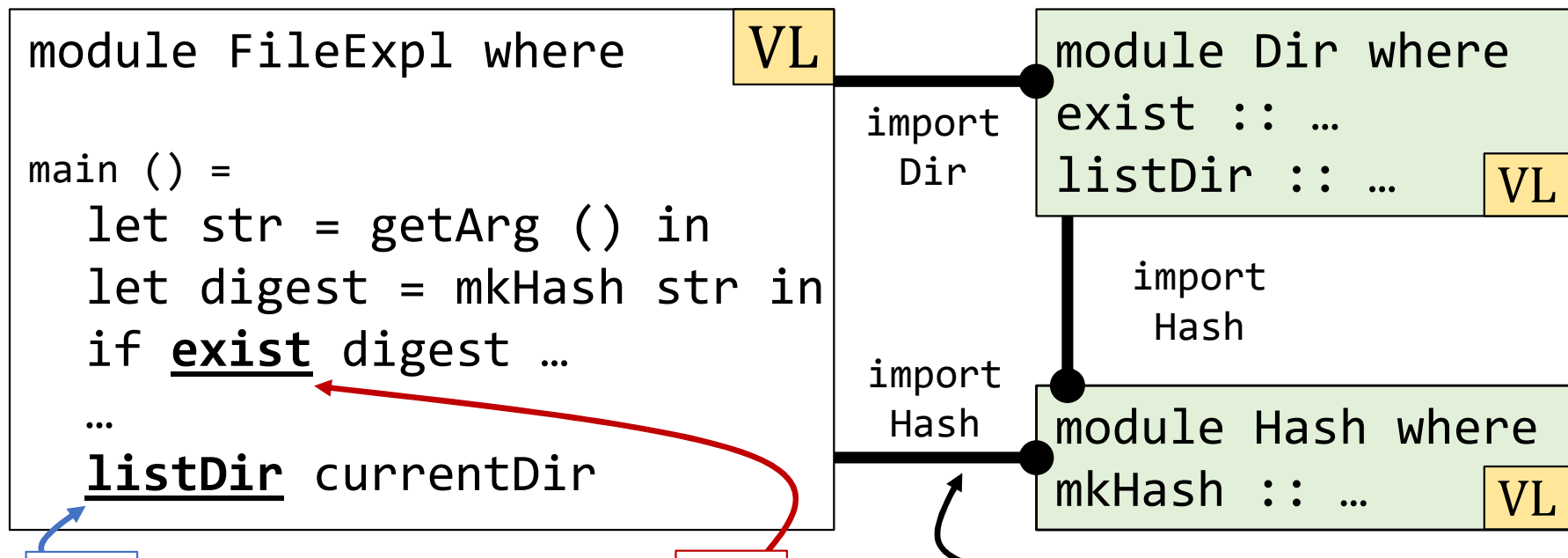


VL vs. λ_{VL} : w/ Version Labels



VL vs. λ_{VL} : w/o Version Labels

No version annotations!



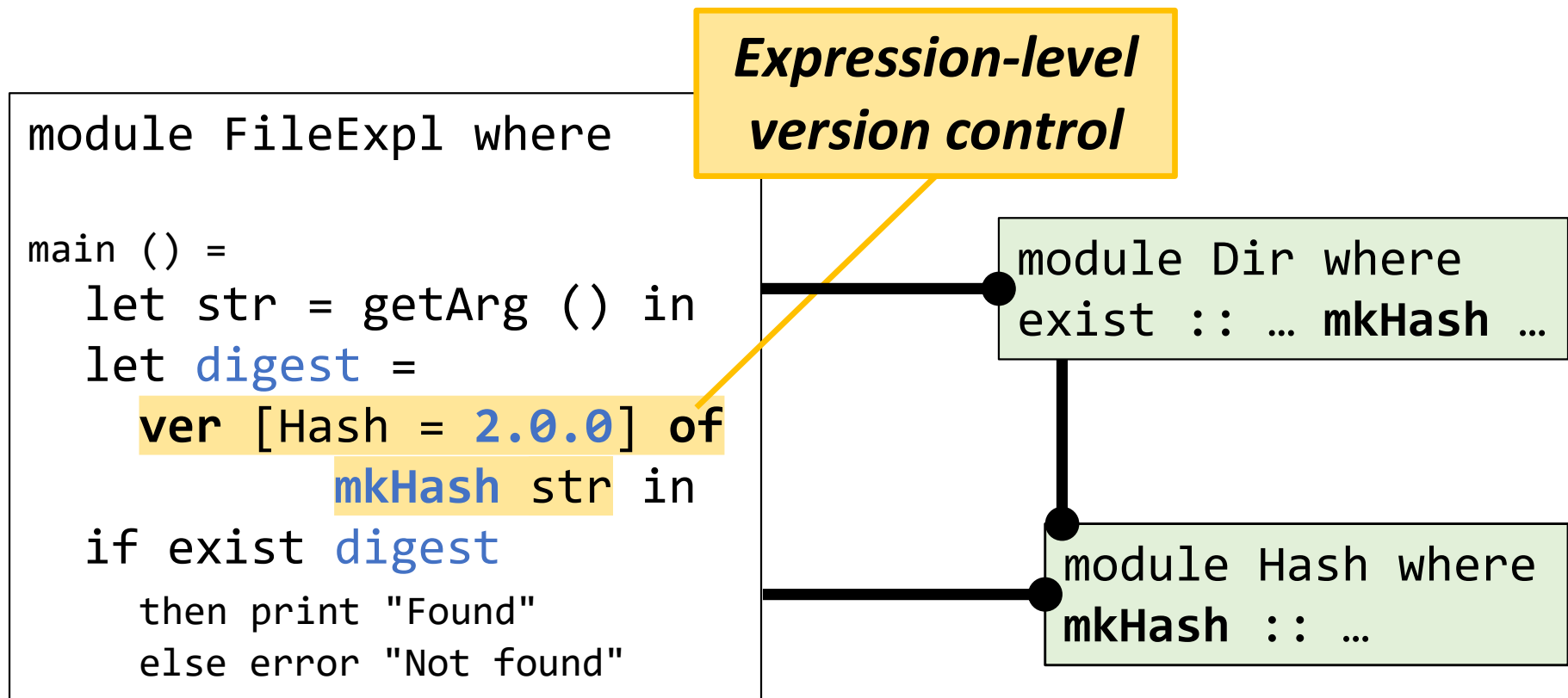
2.0.0

Determine a version
by semantic analysis

1.0.0

Enable refer all versions
through multi-version interfaces

Handling Multiple Versions in One Client



Detecting *Incompatible* Version Usage

```
module FileExp1 where

main () =
  let str = getArg () in
  let digest =
    ver [Hash = 2.0.0] of
      mkHash str in
  if exist digest
  then print "Found"
  else error "Not found"
```

If **exist** depends
1.0.0 for **mkHash** ...

```
module Dir where
exist :: ... mkHash ...
```

```
module Hash where
mkHash :: ...
```

Type checking failed

exist expects an argument from **Hash 1.0.0**,
but **digest** is a value from **Hash 2.0.0**.

Collaboration with *Compatible* Version

```
module FileExp1 where

main () =
  let str = getArg () in
  let digest =
    ver [Dir = 2.0.0] of
      mkHash str in
  let exist' = unver exist
  if exist' digest
  then print "Found"
  else error "Not found"
```

If programmers know **1.0.0** and **2.0.0** of Hash are *compatible* ...

```
module Dir where
  exist :: ... mkHash ...
```

```
module Hash where
  mkHash :: ...
```

Incorporate compatibility into type checking

Outline

Contribution

Programming with Versions *w/o* Version Annotations

[«Programming»'22]

λ_{VL}

Explicit
version annotations

- λ_{VL} Semantics
and Type System

vs.

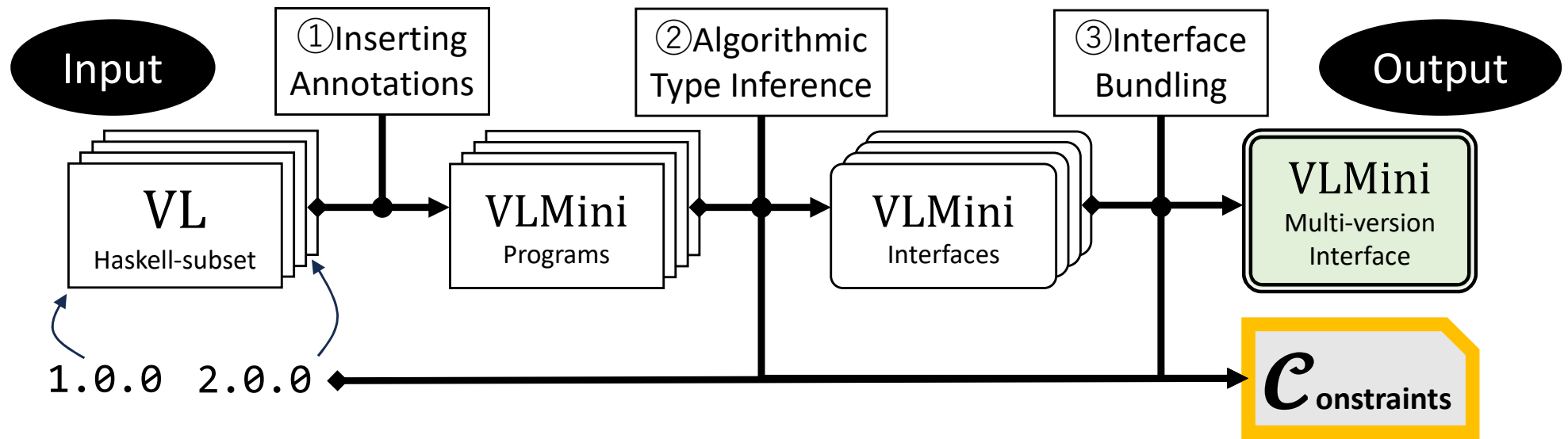
This research

VL

Version inference
incorporating implicit versions

- Key idea:
Utilizing module versions
for expression versions
- VL Programming
- **Compilation Overview**

I/O of Compilation



Next slide 

How the VL compiler uses generated constraints

Compilation Overview

Constraints

“ \leq ” represents dependencies

(Constraints) $\mathcal{C} ::= T \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \mathcal{C}_1 \vee \mathcal{C}_2$

(Dependencies) $\mathcal{D} ::= \langle \overbrace{M_i} \mapsto \overbrace{V_i} \rangle$

Module
name

Version
number

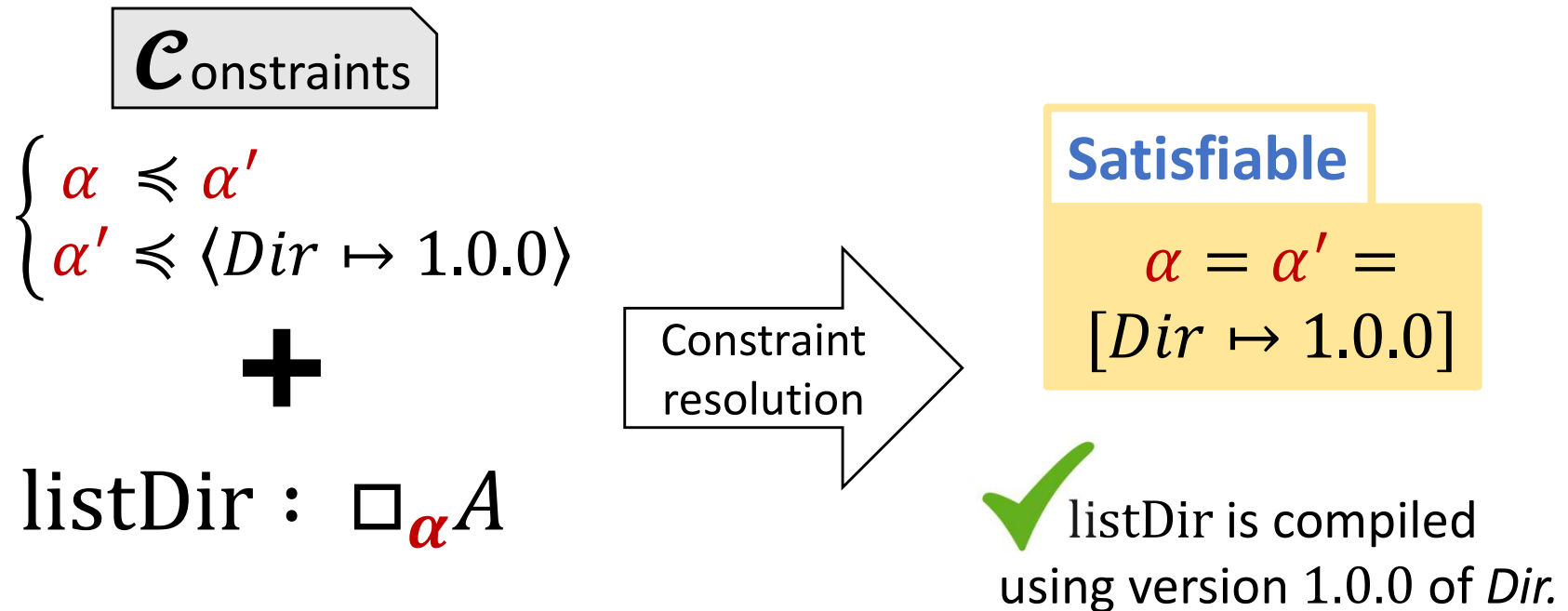
$\alpha \leq \alpha' \mid \alpha \leq \mathcal{D}$

“If a version label for **RHS** expects a specific version, ...

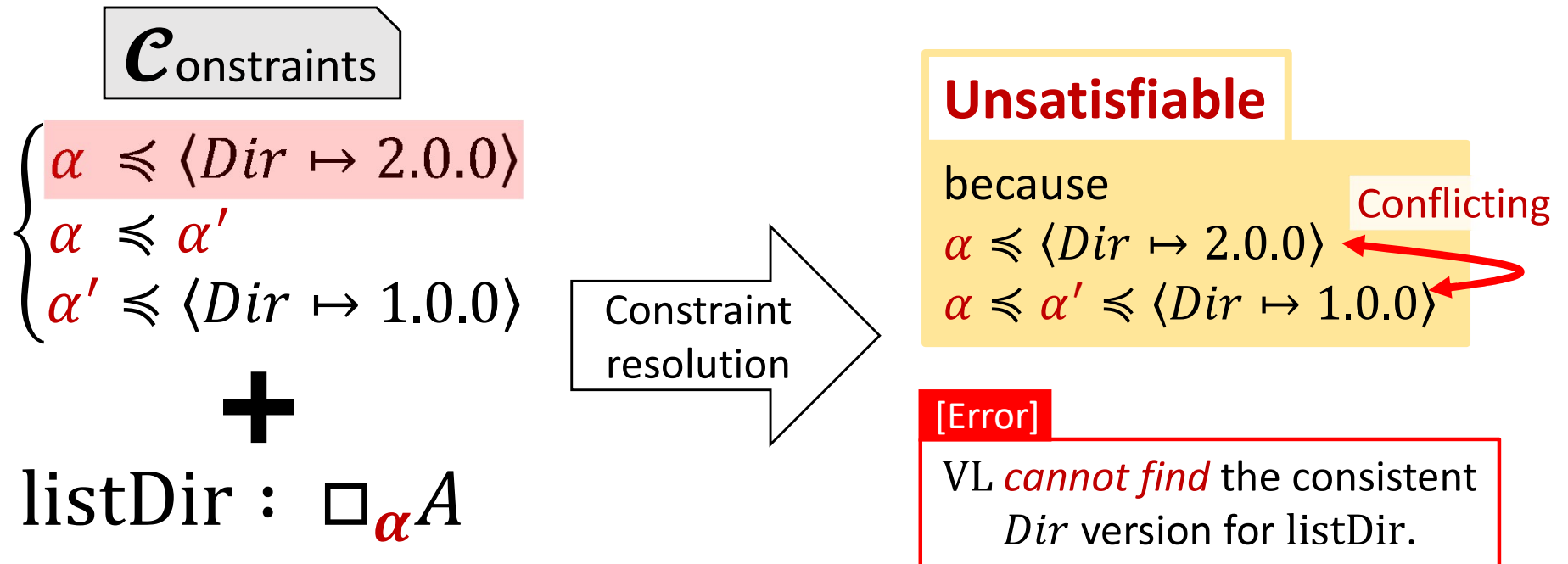
$\alpha \leq \alpha'$ $\alpha \leq \langle Dir \mapsto 1.0.0 \rangle$

... then α (LHS) also expects the same version.”

Satisfiable Constraints



Unsatisfiable Constraints



In the Paper

Proved

Formalization

arXiv:2310.00298

Implementation

<https://github.com/yudaitnb/vl>

Evaluation

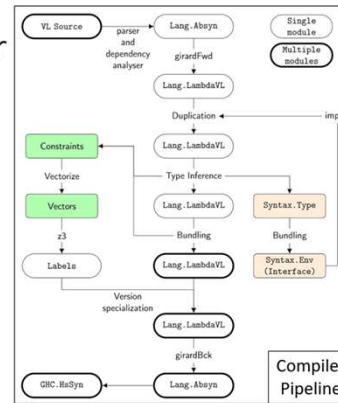
with case studies

- Algorithmic type & version inference
- Proof of soundness wrt λ_{VL} semantics

Implementation

The VL Compiler

- Implemented on GHC 9.2.4 <https://github.com/yudaitnb/vl>
- Both in-/out-put are Haskell ASTs (subset)
- Resolve constraints using Z3 [De Moura'08]



Evaluation

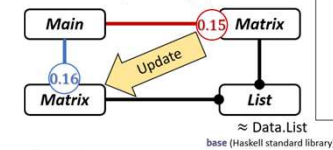
1. Case Study

VL achieves our goals:

- ✓ Handling two versions in one client
- ✓ Detecting inconsistent version

Setting

- Port *hmatrix* to *Matrix*
- Simulating breaking updates in VL



hmatrix: Numeric Linear Algebra

[bsd3,library,math] [Propose Tags]

Linear systems, matrix decompositions, and other numerical computations based on BLAS and LAPACK.

Changelog for hmatrix

0.16.0.0

*join deprecated (use *vjoin).

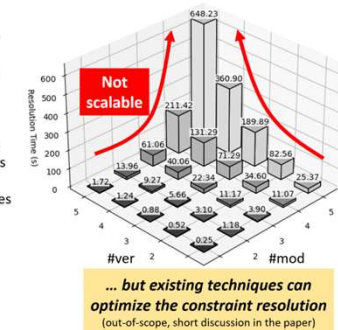
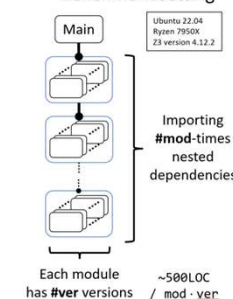
*Added sortVector, roundVector

<https://hackage.haskell.org/package/hmatrix-0.20.2/changelog>

Evaluation

2. Compiler Performance

Benchmark setting



Summary

Contribution

Programming With Versions *w/o* version annotations

[Programming'22]

λ_{VL}


Explicit
version annotations

vs.

This research

VL

Version inference
incorporating implicit versions



Asian Symposium on Programming Languages and Systems
↳ APLAS 2023: **Programming Languages and Systems** pp 3–23 | [Cite as](#)

[Home](#) > [Programming Languages and Systems](#) > Conference paper

Compilation Semantics for a Programming Language with Versions

[Yudai Tanabe](#) , [Luthfan Anshar Lubis](#), [Tomoyuki Aotani](#) & [Hidehiko Masuhara](#)